

Recursion 1

Demo Video :- <https://www.youtube.com/watch?v=-xMYvVr9fd4>

RECURSION :- ① Function calling itself

② A problem is broken down into smaller problem (Subproblem) & the solution is generated using the subproblems

$$\text{Ex:- } \text{Sum}(N) = 1 + 2 + 3 + 4 + \dots \dots + N$$

OR $\text{Sum}(N) = \text{Sum}(N-1) + N$

$$\text{Eg:- } \text{Sum}(10) = \text{Sum}(9) + 10$$

Question :- How to write Recursive Code?

↳ We can solve any recursive code using below Magic steps! -

① Assumption :- Decide what the function does for the given problem

② Main logic :- Break the problem down into smaller subproblems to solve the assumption.

③ Base Case :- Identify the inputs for which we need to stop recursion

FUNCTION CALL TRACING

To understand Recursion i.e how function calls work.

Code :-

```
int add ( int x, int y ) {  
    ✓ return x + y;  
}
```

```
int mul ( int x, int y, int z ) {  
    ✓ return x + y * z;  
}
```

```
int sub ( int x, int y ) {  
    ✓ return x - y;  
}
```

```
void print ( int x ) {  
    ✓ SOUT ( x );  
}
```

```
int main () {  
    ✓ int x = 10;  
    ✓ int y = 20;  
    ✓ t1 = add ( x, y );  
    ✓ t2 = mul ( t1, 75, 1 );  
    ✓ t3 = sub ( t2, 40 );  
    ✓ SOUT ( t3 )  
    PRINT
```

X	PRINT	x = 2210
X	Sub	x = 2250, y = 40
X	Mul	x = 30, y = 75, z = 1
X	Add	x = 10, y = 20
X	Main	x = 10, y = 20 t ₁ = 30, t ₂ = 2250 t ₃ = 2210

Function call stack.

OUTPUT
↳ 2210

Question :- Given a positive integer N , find factorial of N

Ques 1 :- If $N=5$, find the factorial of N

$$5! = 5 \times 4 \times 3 \times 2 \times 1 \\ = 120$$

IDEA :- $\text{fact}(N) = \text{fact}(N-1) * N$

$$\text{Eg:- } \text{fact}(5) = \text{fact}(4) * 5$$

↳ Recursive Code

↳ Let's use 3 magical steps.

PSEUDO CODE

function $\text{fact}(N)$ {

Base Case ←

if ($N == 0$) { return 1; }

Main logic {

int temp = $\text{fact}(N-1)$;

return temp * N

→ Assumption

}

$$3 = 3 \times 2 \times 1 \\ = 6$$

DRY RUN

①st way:- Using Code

function fact (N : 3) {

 if (N == 0) { return 1; }

 int temp = fact (2); // 2

 return temp * N // 2 * 3 = 6

function fact (N = 2) {

 if (N == 0) { return 1; }

 int temp = fact (1); // 1

 return temp * N // 1 * 2 = 2

function fact (N = 1) {

 if (N == 0) { return 1; }

 int temp = fact (0); // 1

 return temp * N // 1

function fact (N = 0) {

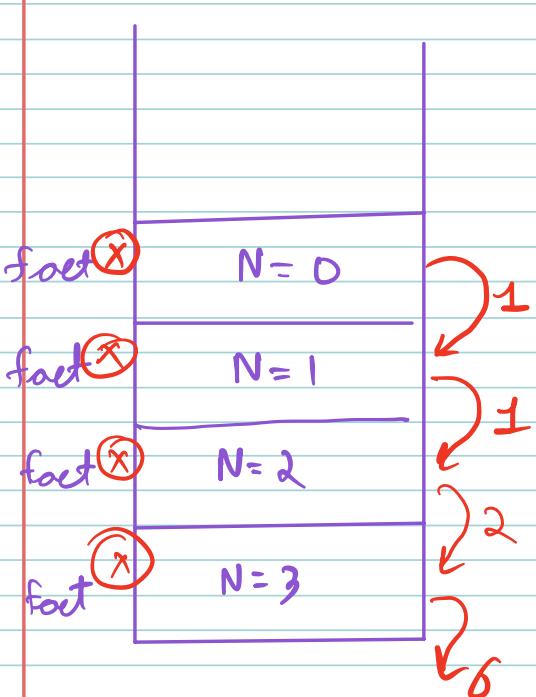
 if (N == 0) { return 1; }

 int temp = fact ();

 return temp * N

3

Way 2:- Using call stack.



```
function fact (N = 3) {
    if (N == 0) { return 1; }
    int temp = fact (N-1);
    return temp * N
}
```

Question :- Fibonacci Series

Eg:-
$$\begin{array}{ccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \dots \\ 0 & 1 & 1 & 2 & 3 & 5 & 8 & 13 & 21 & \dots & \dots \end{array}$$

The Fibonacci sequence is a series of Nos. in which each no is the sum of two preceding numbers.

Ex:-I/P \rightarrow 6th NoO/P \rightarrow 8I/P \rightarrow 8th No.O/P \rightarrow 21

Ques 2 :- If $N=7$, find the N^{th} No in Fibonacci Sequence.

$$\hookrightarrow \boxed{\text{Ans} = 13}$$

IDEA :-

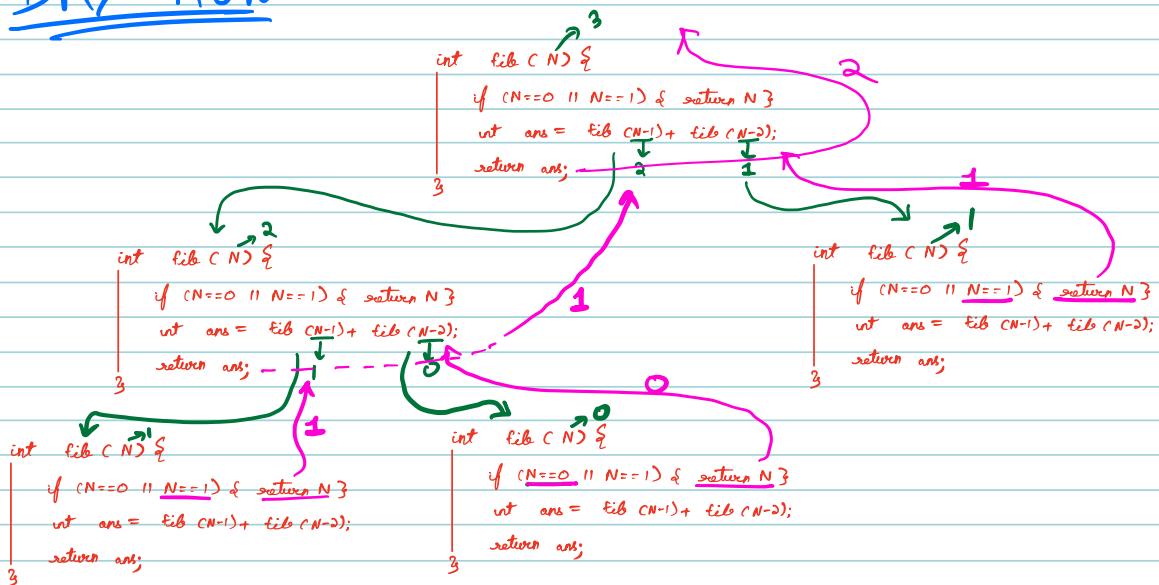
$$\text{fib}(N) = \text{fib}(N-1) + \text{fib}(N-2)$$

$$\begin{aligned}\text{Eg:- } \text{fib}(7) &= \text{fib}(6) + \text{fib}(5) \\ &= 5 + 8 \\ &= 13\end{aligned}$$

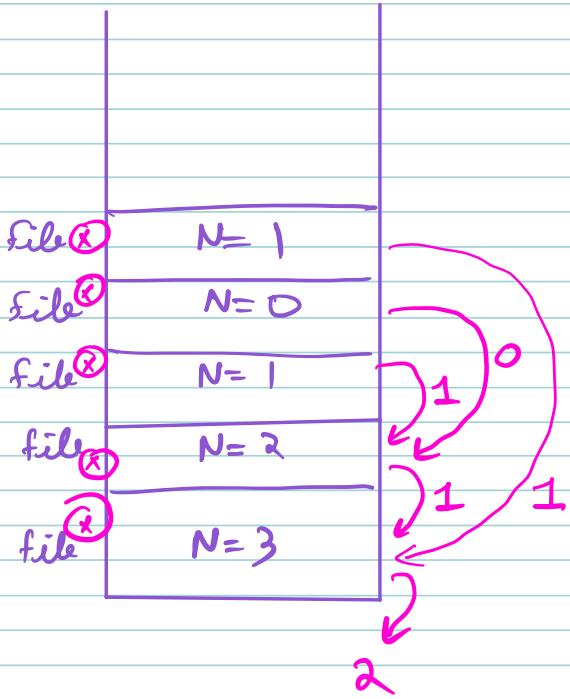
Recursive Code

```
int fib ( N ) {  
    Base Case → if ( N == 0 || N == 1 ) { return N; }  
    Main Logic } { int ans = fib(N-1) + fib(N-2);  
    return ans; } Assumption
```

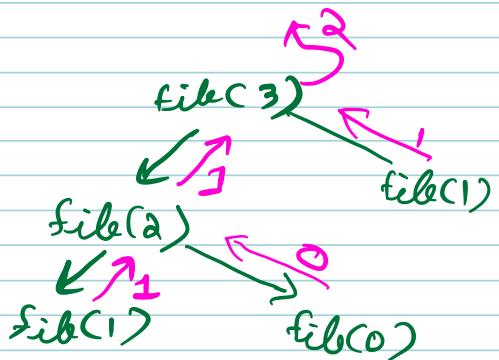
DRY RUN



↳ Using call stack



```
int fib (N) {  
    if (N==0 || N==1) { return N; }  
    int ans = fib (N-1) + fib (N-2);  
    return ans;  
}
```



Question :- Given two integers a & n . Find a^n using recursion.

Ex:- $a = 2, n = 3$

$$2^3 \rightarrow 8 \text{ Ans}$$

IDEA :- $\text{Power}(a, n) = \text{Power}(a, n-1) * a$

Eg:- $2^5 = 2^4 * 2$

PSEUDO CODE

```
fun Power (a, n) {
    if (n == 0) {return 1;}
    return Power (a, n-1) * a;
}
```

DRY RUN

```
Power (2, 3)
      ↓↑4
Power (2, 2)
      ↓↑2
Power (2, 1)
      ↓↑1
Power (2, 0)
```

↳ OPTIMIZATION

current way :-

$$2^4 = 2^3 * 2$$

$$2^{10} = 2^9 * 2$$

Better way :-

$$2^4 = 2^2 * 2^2$$

$$2^{10} = 2^5 * 2^5$$

IDEA

if n is even,

$$\text{Row}(a, N) = \text{Row}(a, \frac{N}{2}) * \text{Row}(a, \frac{N}{2})$$

if n is odd,

$$\text{Row}(a, N) = \text{Row}(a, \frac{N}{2}) * \text{Row}(a, \frac{N}{2}) + a$$

$$\hookrightarrow 2^n = 2^5 * 2^5 + a$$

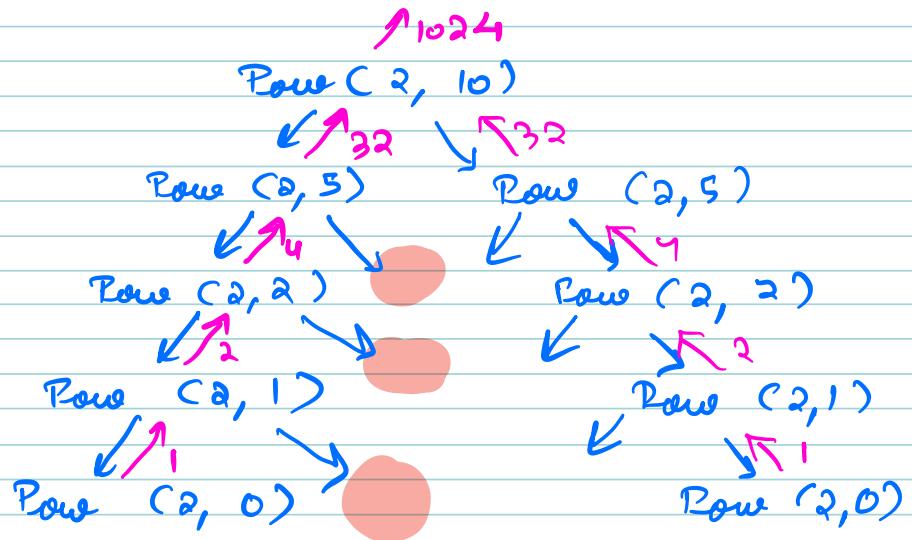
PSEUDO CODE

```

func Row(a, n) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        return Row(a, n/2) * Row(a, n/2);
    } else {
        return Row(a, n/2) * Row(a, n/2) + a;
    }
}

```

DRY RUN



Stupidity :- Redundant call for the value which is already computed.

PSEUDO CODE [East Exponentiation / East Power]

```
func Pow (a, n) {
    if (n == 0) return 1;
    int temp = Pow (a,  $\frac{n}{2}$ );
    if (n % 2 == 0) {
        return temp * temp;
    } else {
        return temp * temp * a;
    }
}
```

DRY RUN

Rowe (2, 10)
↓↑32
Rowe (2, 5)
↓↑4
Rowe (2, 2)
↓↑2
Rowe (2, 1)
↓↑1
Rowe (2, 0)

TIME COMPLEXITY

① PSEUDO CODE

```
function fact (N) {  
    if (N == 0) { return 1; }  
    int temp = fact (N-1);  
    return temp * N  
}
```

Recurrence Relation

$$T(N) = T(N-1) + 1$$

For some extra
done on assumption

Base Case

$$T(0) = 1$$

↳ Solving Recurrence Relation

(Maths)

$$T(N) = T(N-1) + 1$$

$$\downarrow \quad T(N-1) = T(N-2) + 1$$

$$\Rightarrow T(N) = [T(N-2) + 1] + 1$$

$$\Rightarrow T(N) = \overline{T(N-2)} + 2$$

⇒

$$\downarrow \quad T(N-2) = T(N-3) + 1$$

$$\Rightarrow T(N) = T(N-3) + 3$$

Generic

$$T(N) = T(N-k) + k$$

$$\text{Goal, } T(N-k) = T(0)$$

$$\hookrightarrow N-k = 0$$

$$\hookrightarrow \boxed{N = k}$$

←

$$\Rightarrow T(N) = T(N-N) + N \rightarrow$$

$$= T(0) + N$$

$$\boxed{T(N) = 1 + N} \quad - \quad \boxed{\text{Base Case}}$$

$$T.C. = O(N)$$

(2)

PSEUDO CODE

```
func Pow ( a, n ) {  
    if ( n == 0 ) return 1;  
    if ( n % 2 == 0 ) {  
        return Pow ( a,  $\frac{n}{2}$  ) * Pow ( a,  $\frac{n}{2}$  );  
    } else {  
        return Pow ( a,  $\frac{n-1}{2}$  ) * Pow ( a,  $\frac{n-1}{2}$  ) * a;  
    }  
}
```

$T\left(\frac{N}{2}\right)$

Recurrence Relation

$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + 1$$

$$\Rightarrow T(N) = 2T\left(\frac{N}{2}\right) + 1$$

$$T(0) = 1, \quad T(1) = 1$$

maths :-

$$T(N) = 2T\left(\frac{N}{2}\right) + 1$$

$$\downarrow T\left(\frac{N}{2}\right) = 2T\left(\frac{N}{4}\right) + 1$$

$$\Rightarrow T(N) = 4T\left(\frac{N}{4}\right) + 3$$

$$\downarrow T\left(\frac{N}{4}\right) = 2T\left(\frac{N}{8}\right) + 1$$

$$\Rightarrow T(N) = 4 \left[2T\left(\frac{N}{8}\right) + 1 \right] + 3$$

$$\Rightarrow T(N) = 8T\left(\frac{N}{8}\right) + 7$$

$$\xrightarrow{\text{Let}} T\left(\frac{N}{8}\right) = 2T\left(\frac{N}{16}\right) + 1$$

$$\Rightarrow T(N) = 16T\left(\frac{N}{16}\right) + 15$$

Generic

$$T(N) = \left[2^k T\left(\frac{N}{2^k}\right) \right] + (2^k - 1)$$

AIM:- Remove T from right

$$\textcircled{X} \quad \frac{N}{2^k} = 0 \quad \Rightarrow \quad N = 0$$

$$\Rightarrow \frac{N}{2^k} = 1 \quad \Rightarrow \quad N = 2^k \quad \Rightarrow \quad k = \log N$$

$$\Rightarrow T(N) = 2^{\log N} T\left(\frac{N}{2^{\log N}}\right) + (2^{\log N} - 1)$$

$$\Rightarrow T(N) = N T\left(\frac{N}{N}\right) + (N-1) \quad \boxed{a^{\log a b} = b}$$

$$= N T(1) + (N-1)$$

$$= \cancel{N+1} + (N-1)$$

$$\Rightarrow T(N) = 2N - 1$$

T.C. $\Rightarrow O(N)$

③ PSEUDO CODE [FAST POWER]

```
func Pow ( a, n ) {  
    if ( n == 0 ) return 1;  
  
    int temp = Pow ( a,  $\frac{n}{2}$  );  
    if ( n % 2 == 0 )  
        return temp * temp;  
    else {  
        }  
        return temp * temp * a  
    }  
}
```

Recurrence Relation

$$T(N) = T\left(\frac{N}{2}\right) + 1$$

Base Case

$$T(1) = 1$$

Maths :- $T(N) = T\left(\frac{N}{2}\right) + 1$

$$\downarrow$$
$$T\left(\frac{N}{2}\right) = T\left(\frac{N}{4}\right) + 1$$

$$\Rightarrow T(N) = T\left(\frac{N}{4}\right) + 2$$
$$\downarrow$$
$$T\left(\frac{N}{4}\right) = T\left(\frac{N}{8}\right) + 1$$

$$\Rightarrow T(N) = T\left(\frac{N}{8}\right) + 3$$

⋮

Generic

$$\Rightarrow T(N) = T\left(\frac{N}{2^K}\right) + K$$

$$\Leftrightarrow \frac{N}{2^K} = 1$$

$$\Rightarrow N = 2^K \Rightarrow K = \log_2 N$$

$$\Rightarrow T(N) = T\left(\frac{N}{2^{\log_2 N}}\right) + \log_2 N$$

$$= T\left(\frac{N}{N}\right) + \log_2 N$$

$$= T(1) + \log_2 N$$

$$\Rightarrow T(N) = 1 + \log_2 N$$

$$TC \rightarrow O(\log_2 N)$$

Q) Fibonacci

```
int fib ( N ) {  
    if ( N == 0 || N == 1 ) { return N }  
    int ans = fib ( N - 1 ) + fib ( N - 2 );  
    return ans;  
}
```

Recurrence Relation

$$T(N) = T(N-1) + T(N-2) + 1$$

↳ Directly this will not simplify to general relation

$$\because T(N-1) > T(N-2)$$

So, $T(N-2)$ we can take $T(N-1)$ instead of its something extra not less.

$$\Rightarrow T(N) = 2T(N-1) + 1$$

↳ P.L.W.

$$T \cdot C \rightarrow O(2^n)$$

ANOTHER WAY OF CALCULATING T.C.

Time complexity:- Time taken by one function call * No. of function calls

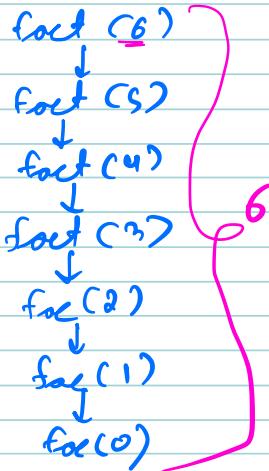
Ques 3 :- How many recursive calls in the factorial(6)?

function fact(N) {

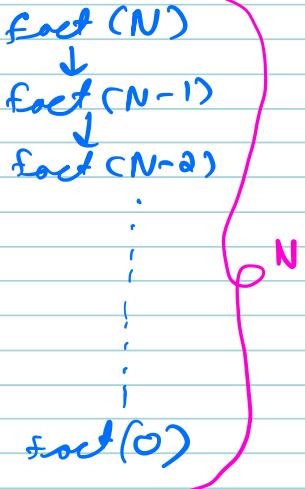
 if (N == 0) { return 1; }

 int temp = fact(N-1);

 return temp * N



Generic



$$TC \rightarrow 1 * N \leftarrow \text{Function call}$$

Time taken by Function

$$TC \rightarrow O(N)$$

Ques 4

How many recursive call in the FAST pow(2,5)

Func Pow2(a, n) {

 if (n == 0) { return 1; }

 temp = Pow2(a, n/2);

 if (n%2 == 0) {

 return temp * temp;

 } else {

 return temp * temp * a;

}

Pow(2,5)

↓

Pow(2,2)

↓

Pow(2,1)

↓

Pow(2,0)

Generic

Pow(q,N)

↓

Pow(q, N/2)

↓

Pow(q, N/4)

↓

⋮

Pow(q,0)

$\log_2 N$

$$TC \rightarrow 1 * \log_2 N$$

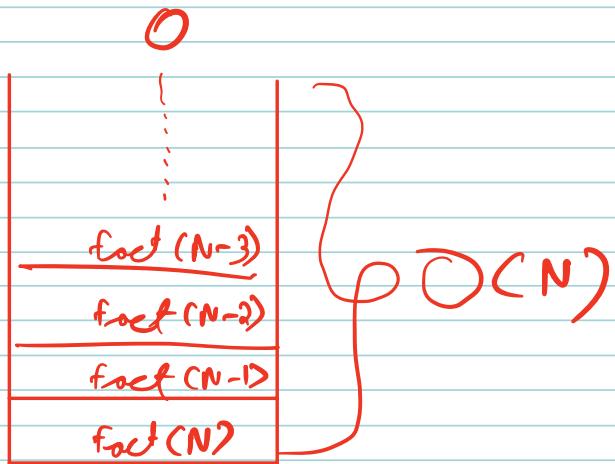
$\log_2 N$

T.C. $\rightarrow O(\log_2 N)$

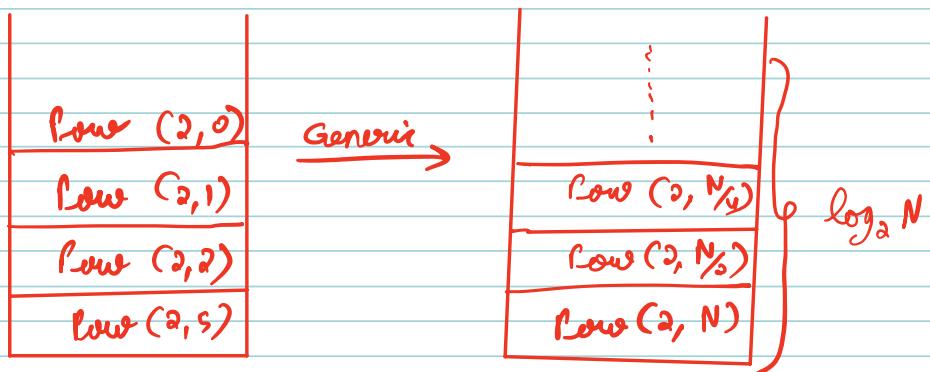
SPACE COMPLEXITY of Recursive Call

↳ maximum function in call stack at one time.

① For Factorial



② For Power

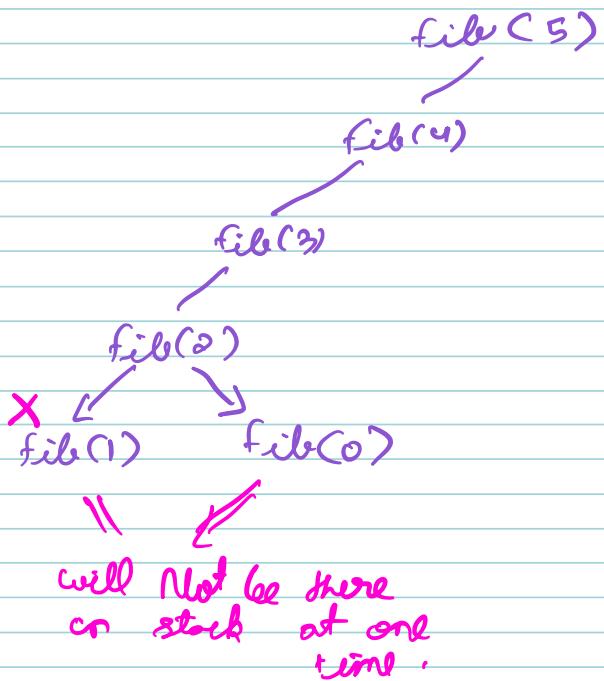


$$SC \rightarrow O(\log_2 N)$$

③

Fibonacci

```
int fib ( N ) {  
    if ( N == 0 || N == 1 ) { return N ; }  
    int ans = fib ( N - 1 ) + fib ( N - 2 );  
    return ans;  
}
```



$SC \rightarrow O(N)$